Serene: Self-Reliant Client-Side Protection against Session Fixation

Philippe Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, Wouter Joosen

SERENE: Self-Reliant Client-Side Protection against Session Fixation*

Philippe De Ryck, Nick Nikiforakis, Lieven Desmet,
Frank Piessens, and Wouter Joosen

IBBT-DistriNet, KU Leuven, 3001 Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract The web is the most wide-spread and de facto distributed platform, with a plethora of valuable applications and services. Building stateful services on the web requires a session mechanism that keeps track of server-side session state, such as authentication data. These sessions are an attractive attacker target, since taking over an authenticated session fully compromises the user's account. This paper focuses on session fixation, where an attacker forces the user to use the attacker's session, allowing the attacker to take over the session after authentication.
We present SERENE, a self-reliant client-side countermeasure that protects the user from session fixation attacks, regardless of the security provisions – or lack thereof – of a web application. By specifically protecting session identifiers from fixation and not interfering with other cookies or parameters, SERENE is able to autonomously protect a large majority of web applications, without being disruptive towards legitimate functionality. We experimentally validate these claims with a large scale study of Alexa's top one million sites, illustrating both SERENE's large coverage (83.43%) and compatibility (95.55%).

keywords: web applications, security, session fixation.

1 Introduction

In the past few years, the security community has witnessed a shift in attacks originating from malicious individuals and the organized criminal underground. Attacks usually targeting the server-side of the Internet (e.g. Web, Mail and FTP servers) are now conducted on the client-side, targeting the site, the user's browser or even the user himself. This phenomenon can be ascribed to the enormous expansion of web sites and web applications, which currently almost monopolize a user's online activities. A substantial fraction of these attacks targets a web application's session management, the cornerstone of any stateful web application. Session management enables building stateful applications on top

* This work incorporates contributions from KU Leuven master students Bram Bonné [4] and Joeri Ledegen. This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT, IWT, the Research Fund KU Leuven and the EU-funded FP7-projects WebSand and NESSoS.

of a stateless protocol (HTTP), by grouping multiple related requests together into a session. Each session is assigned a unique identifier and can keep track of session-specific data, such as preferences, user information or authentication state. Sessions are typically maintained by cookies, part of the HTTP headers, or parameters, embedded in the content.

One well-known session attack is *session fixation*. In a session fixation attack, the attacker establishes a session between him and the target application, and subsequently forces this session into the user's browser. Any action taken by the user within the application is associated with the user's session, which is in this case identical to the attacker's session. For example, if the user authenticates herself to the application, the session remembers the user's information and authentication state. In case of a session fixation attack, the attacker shares the same session, allowing him to perform actions in the user's name. Session fixation is ranked third in the OWASP top 10 of web application security risks, and is assigned a prevalence of *common* [18].

An adequate, widely available by-design mitigation technique for session fixation is to issue a new (thus non-fixated) session identifier whenever the privilege level of a user changes, for example from unauthenticated to authenticated. Unfortunately, studies have shown that security guidelines are not applied as widespread as one would hope or expect [15,19], thus leaving the user vulnerable for potential session fixation attacks.

We present SERENE, a self-reliant client-side countermeasure against session fixation attacks. SERENE is compatible with applications using both cookie-based and/or parameter-based session management. The main idea behind SERENE is to prevent the browser from sending fixated session identifiers through cookies, and to prevent the use of fixated session identifiers through parameters embedded in the pages' contents. To distinguish session identifiers from other cookies or parameters, we present an elementary algorithm that supports a large majority of sites, but still maintains a very low false positive rate. To validate our identification algorithm and test our prototype implementation, we conducted a large scale study of Alexa's top one million sites, showing both the wide range of support and the compatibility of SERENE.

The remainder of this paper is organized as follows: Section 2 introduces parameter-based and cookie-based session management techniques. Section 3 focuses entirely on session fixation, including different attack vectors, current countermeasures and a real-life example attack scenario. Section 4 presents SERENE, our client-side countermeasure against session fixation attacks. We extensively validate SERENE using the Alexa top one million (Section 5). Finally, we discuss difficulties with re-using existing previous work, as well as potential improvements (Section 6). We conclude the paper in Section 7.

2 Session Management

Virtually every non-static web application embodies stateful behavior such as identifying individual users, enforcing access control rules and distinguishing

simultaneously submitted requests. Due to the stateless nature of HTTP, this stateful behavior is enabled on top of HTTP by introducing sessions. Sessions link multiple requests from the same client together and allow stateful information to be accessed and updated during the course of that session.

The de facto implementation of sessions consists of a server-side stored session state, for which the server generates a random, unique session identifier (SID) [13]. The client is instructed to include the assigned SID with every request, allowing the server to link multiple requests from this client to the same session. There are two common ways a web application can instruct the client to include a SID: cookies and parameters. We discuss both approaches separately in the remainder of this section.

2.1 Cookies as Session Identifiers

Cookies are key/value pairs belonging to the domain that sets them, potentially extended by certain options (e.g. *Path*, *Secure*, etc.). The browser keeps track of cookies in the so-called cookie jar. When the browser sends a request to a certain domain, it attaches all known cookies for that domain using the `Cookie` request header. Traditionally, cookies are set by the server using either the `Set-Cookie` response header, by embedding a `Set-Cookie` meta tag in the body or by including JavaScript that sets a cookie when executed by the browser.

Cookies typically belong to the domain that sets them (e.g. `www.example.com`), but using the *Domain* option, they can also be bound to a parent domain (e.g. `.example.com`), in which case they belong to all subdomains of `example.com`. This feature is often used to share the same cookies among different parts of an application (e.g. `login.bank.com` and `payments.bank.com`). Setting the *Domain* option to a top-level domain (e.g. `.com`) is not allowed.

Implementing session management using cookies is straightforward: a session is typically created by the server after receiving the first (cookieless) request, and the generated SID is attached as a cookie to the response. The browser stores the cookie containing the SID and attaches it to every request going to this specific domain. Upon receiving a request containing a cookie with a SID, the server can link the request to the associated session. A new SID can easily be assigned by sending the client a new cookie with the same name but a different value, which overwrites the old value.

2.2 Parameters as Session Identifiers

Since not all clients support cookies, or cookie support can be explicitly disabled by the user, an alternative approach is to include the SID as a parameter in every request to the server. Examples are to include the SID as a parameter in the URL of a link, or as a hidden field in a form element. Maintaining a session this way requires the server to ensure that all URLs pointing to its own domain contain the SID of the associated session. When the user opens a URL with an embedded SID (e.g. by clicking on a link), the browser sends a request containing

the embedded SID, allowing the server to extract the SID and link the request to the associated session.

Popular web frameworks offer embedded support for session management, which includes both cookie-based and parameter-based session management. Sites running on top of such a framework can easily support the parameter-based fallback mode if desired.

2.3 Attacks on Session Management

Attacks on session management are popular, since they offer a high reward for the attacker. Successful attacks can involve the attacker making specific requests in the user's name, or an attacker having full control over the user's session, allowing him to access all information and perform all actions available to the user. Concrete attack examples include session hijacking and cross-site request forgery (CSRF). Existing work proposes specific client-side countermeasures to prevent both session hijacking [9,13] and CSRF attacks [6,14].

In this paper we focus on session fixation and propose a client-side countermeasure against this attack. To the best of our knowledge, SERENE is the first concrete proposal for client-side protection against session fixation attacks.

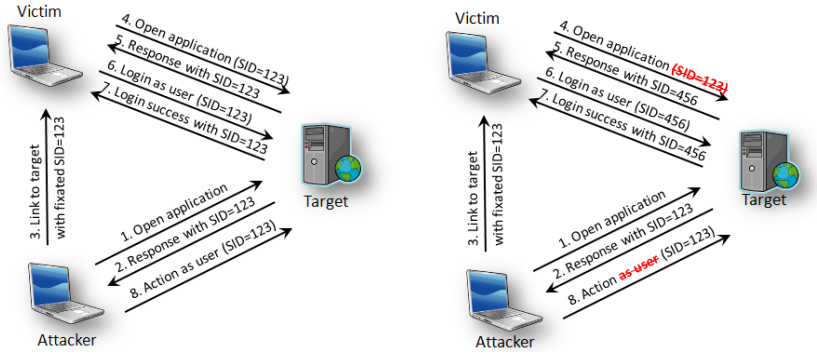
3 Session Fixation

The goal of a session fixation attack is to gain control over a session of an authenticated user, thus giving the attacker full access to the target application with the user's privileges. To reach this goal, the attacker will force the user to use a session accessible to the attacker, by fixating a known SID in the user's browser before authentication. When the user has successfully authenticated, the attacker can contact the target application with the same SID, thus impersonating the authenticated user. Figure 1(a) shows a session fixation attack on a parameter-based session management system.

To launch a session fixation attack against a vulnerable application, the attacker needs to be able to fixate the SID in the user's browser. The attacker can gain this capability by launching the attack from a web site, the so-called web attacker [2], or through an out-of-band channel, such as email or instant messaging. In the remainder of this section, we discuss ways for an attacker to perform session fixation attacks on parameter-based and cookie-based session management systems. Additionally, we discuss currently available protection mechanisms against session fixation attacks, followed by a real-life attack scenario discovered during the writing of this paper.

3.1 Parameter-based Session Fixation

A1. Links The simplest attack vector is to get the user to go to the target site through a crafted link, that contains a SID embedded as a URL parameter (e.g. `<a href="http://target.example.com?SESSID=A1B2C3D4E5">Follow me</a>`).



(a) Session Fixation Attack (b) SERENE's General Approach

Figure 1. Schematic representation of a session fixation attack (left) and SERENE's general protection approach (right)

There are numerous ways to trick the user into following a crafted link. One example are external channels, such as email messages or instant messaging, where a user is simply asked to open a link. An attacker can also place a link on an unrelated site, such as an attacker-controlled site or a site permitting user content to be posted, such as a social networking site, a forum, etc. Finally, an attacker can also inject a link containing a SID directly in the target site, making it blend in with the rest of the page contents.

A2. Script Execution If the attacker can execute a malicious script within the origin of the target site, he can easily launch a session fixation attack by replacing the valid embedded SIDs with fixated SIDs.

Script execution privileges can be gained legitimately (e.g. an included advertisement), or unintentionally (e.g. through cross-site scripting (XSS)). Unfortunately, vulnerabilities giving attackers script execution privileges are very common, as indicated by the high-ranked spot in both the OWASP top 10 web application security risks [18]. Additionally, attackers can legitimately gain script execution privileges in numerous ways [1,17].

3.2 Cookie-based Session Fixation

A3. Script Execution Similar to parameter-based session fixation (attack vector *A2*), cookie-based session management is susceptible to session fixation if the attacker gains script execution privileges. The attacker can simply set a fixated SID as a cookie through the `document.cookie` property. Note that a site with an XSS vulnerability that allows cookie-based session fixation, is not necessarily susceptible to other attacks, such as session hijacking [19].

A4. Meta-tags The `http-equiv` attribute supported by HTML meta tags enables several header-like instructions, such as setting cookies. Placing the following code in an HTML page results in the creation of a cookie with name *foo*

and value *bar*: `<meta http-equiv="Set-Cookie" content="foo=bar; Path=/;"/>`. By injecting such a tag in the target site, an attacker can easily fixate a SID in a cookie.

Meta tags are typically included in the header of a page, but an investigation of major browsers shows that meta tags found in the page's body are also honored. Additionally, some browsers also process dynamically included meta tags (e.g. from JavaScript using the `appendChild` operation). Similar to script execution, this attack vector can be exploited through legitimate means (e.g. an included advertisement) or through an injection vulnerability. Note that for meta tag injection, it suffices that the injection vulnerability allows the injection of an HTML tag. A full-scale cross-site scripting vulnerability is not required.

A5. Headers The possibility for an attacker to inject headers into the HTTP response allows him to use the `Set-Cookie` header to fixate a cookie-based SID. Header injection [10] is typically caused by a target site including unsanitized input in header values, or by a parsing vulnerability in a browser or proxy system. Due to the large-scale impact of such a vulnerability in a web framework, language or browser, these vulnerabilities are typically immediately fixed, making header injection an unlikely attack vector.

A6. Subdomains As discussed before, cookies set from a subdomain can apply to other subdomains as well, using the *Domain* option. This feature becomes problematic when not all subdomains belong to the same entity, and the attacker controls one of them.

An obvious way for the attacker to attack a target site sharing the same parent domain is to set a cookie for the parent domain through an HTTP header. If the attacker fully controls this application, setting cookies through HTTP headers is trivial. In cases where the attacker has no control over the headers (e.g. limited hosting with only static pages), he can achieve the same goal by including a meta tag in one of his pages or by setting a cookie from JavaScript through the `document.cookie` property. At the end of this section, we discuss a real-life session fixation attack where the attacker is able to execute scripts in a subdomain, without having control over the headers.

3.3 Current Countermeasures

Session fixation attacks have been known for some time, and adequate server-side protection techniques are available. Unfortunately, these protection techniques are not always deployed, leaving the user vulnerable to session fixation attacks. We discuss the most important and effective countermeasures below.

Session fixation can easily be addressed during the development phase of a web application, by generating a new session identifier whenever the privilege level of a user changes, for example from an unauthenticated state to an authenticated state. This approach foils any session fixation attacks aimed at obtaining an authenticated session. Even if an attacker forces a session identifier on a user,

the session identifier will be overwritten by a newly generated one after authentication. Since the new SID differs from the fixated one, the authenticated user is never linked to the fixated session. This approach works both for cookie-based and parameter-based session management. Most web frameworks explicitly support the regeneration of SIDs, but require the developer to enable it or explicitly trigger it by calling a function.

Instead of focusing on the value of the SID, several approaches aim to generally protect cookies. One example is the *HttpOnly* option that can be added to a cookie, preventing JavaScript from reading that cookie [19], foiling traditional session hijacking attacks. Recently, browsers started preventing an HttpOnly cookie to be overwritten from JavaScript, thus severely limiting the window of opportunity for a session fixation attack using attack vectors *A3*, *A4* and *A5* (i.e. fixation can only happen before the user received a SID from the server). As indicated by other studies, the use of HttpOnly is still fairly limited [15,19].

Bortz et al. [5] propose to limit cookies to their origin, thus preventing the sharing of cookies across subdomains. Origin cookies can effectively prevent session fixation attacks from subdomains, if both browsers and applications implement and deploy this feature.

Furthermore, Johns et al. [8] have proposed two more server-side solutions for combating session fixation attacks against cookie-based session management. One consists of instrumenting the underlying web framework, to automatically regenerate the SID when an authentication process is detected. In a second approach, they propose a server-side proxy that maintains its own SID and couples it to the target site's SID. Renewing the proxy SID after a detected authentication process prevents session fixation attacks. Both approaches do not require modifications to either the web framework or the protected site, but depend on initial training or configuration to identify the authentication process.

3.4 Example Attack Scenario

While researching possible attacks and defenses connected with session fixation, we encountered a two-step session fixation attack on Weebly, a Web 2.0 site builder boasting a userbase of more than 8 million people. When registering on `www.weebly.com`, a dedicated subdomain is assigned to the user (e.g. `alice.weebly.com`). The user can subsequently create her site using a combination of drag-and-drop elements as well as custom HTML, which allows a user to write arbitrary HTML and JavaScript code on her page.

Weebly's cookie configuration settings do not allow a page on a subdomain to steal cookies from sibling domains or the main `www.weebly.com` domain. This effectively prevents session hijacking attacks that attempt to steal cookie-based SIDs through JavaScript. As discussed in attack vector *A6*, JavaScript is allowed to create cookies with the `domain` option set to `.weebly.com`. This operates as a subdomain-wildcard, instructing the browser to send this cookie to all subdomains of `weebly.com`. Thus, an attacker can now set a cookie that will be sent to `www.weebly.com` when the user visits Weebly's home page.

At the same time, we noticed that if a user presents a session identifier to Weebly's login screen and successfully logs-in, Weebly maintains the same identifier. These two "features" provide all the necessary ingredients for a session fixation attack. If a user is lured into visiting a malicious site hosted on a subdomain of `weebly.com`, the attacker can fixate a cookie-based SID valid on `www.weebly.com` in the user's browser. Once the user authenticates to Weebly, the attacker can take full control over the user's session. All the attacker needs to do, is to poll one of Weebly's authenticated pages until he is recognized as the logged-in victim.

We have reported the vulnerability and way of exploitation to Weebly's staff, who verified and corrected the issue as of November 2011. When a Weebly user now authenticates on `www.weebly.com`, the SID will be renewed.

4 Client-Side Protection against Session Fixation

Even though adequate protection techniques and countermeasures exist, web applications do not implement them, leaving the user vulnerable. In this section, we present SERENE, a client-side countermeasure against session fixation, that will protect the user against session fixation attacks, regardless of the security precautions taken by any target application. SERENE does not depend on the user to make security decisions, since the user can not be expected to have the expertise nor the time to make a decision for each application or request.

We present our countermeasure in several steps. First, we discuss the general idea to protect against session fixation attacks. Next, we elaborate on a few peculiarities with parameter-based session management, followed by our algorithm to identify SIDs from the collection of cookie and parameter key/value pairs. Finally, we discuss the prototype implementation as a Firefox add-on.

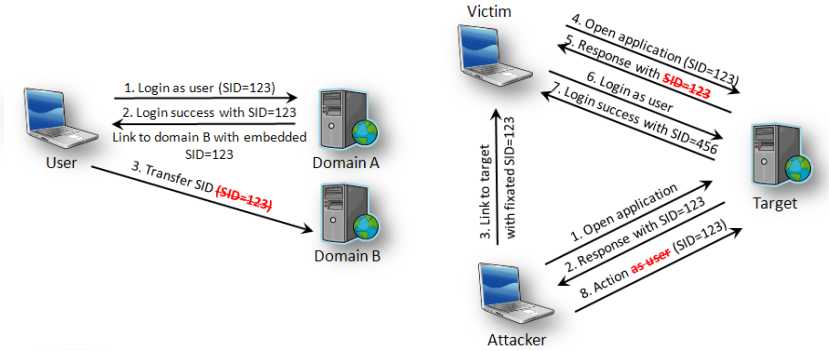
4.1 General Approach

The general approach of protecting against session fixation attacks is to prevent potentially fixated SIDs from being sent to the server. If a fixated SID never reaches the server, session fixation is simply not possible. This general approach, as depicted in Figure 1(b), is applicable for both cookie-based and parameter-based session management.

For cookie-based session management, SERENE keeps track of all legitimately set cookies that contain a SID in an internal database. These cookies are found in the `Set-Cookie` header, part of the response. Next, SERENE scans each outgoing request for attached cookies, by investigating the `Cookie` header. Any attached cookies that contain a SID must also appear in the internal database of legitimate SIDs for the target site. In case a cookie containing a SID is not present in the internal database, it is stripped from the outgoing request. The reasoning behind this approach builds on the fact that it is very unlikely for a server-generated session identifier to be set in other ways than through a header. In Section 5, we offer substantial experimental evidence to support this claim.

Mapping the attack vectors discussed earlier to this approach shows that cookies containing a SID set through JavaScript or meta tags are not known in the internal database, and thus not sent to the server. This holds both for the single domains as well as subdomain case. The only remaining attack vector is through header injection in a single domain, a highly unlikely scenario, or through header injection from a subdomain, discussed in Section 6.

To protect applications with parameter-based session management, incoming pages are scanned for embedded SIDs (e.g. in URLs or as hidden form fields) that identify a session within the domain sending the response. Similar to cookies, these SIDs are stored in the internal database. Outgoing requests are also scanned for any embedded SIDs (e.g. in the URL or in the POST body), and SIDs not present in the database are removed from the request.



(a) Legitimate Cross-Domain Scenario (b) Serene's Parameter Approach

Figure 2. Schematic representation of the general approach mistakenly detecting a cross-domain parameter (left) and Serene's modified approach to allow such tokens to be sent once, but not return in the response (right)

The attack vectors described earlier are foiled by this approach. An externally injected link will contain a SID that is unknown to Serene, so it will be removed. Similarly, an attacker page embedding a SID in a URL to the target domain is not a logical scenario: a SID should be set by the domain that wants to maintain a session, not by another domain. Therefore, such SIDs are not stored in the database, so attaching it to a request will not be allowed by Serene. Additionally, SIDs injected by a script will not yet be present when Serene examines the response, so they will not be stored in the database, therefore they will not be allowed on an outgoing request. Finally, if the attacker injects HTML code containing a fixated SID directly in the target application, it will collide with SIDs added by the application. Serene detects such collisions and either uses a previously stored SID, or removes all offending SIDs from the page.

4.2 Parameter-Based Exchanges

Complex web applications often share SIDs or SID-like tokens across domains using cross-domain requests with embedded parameters. Common examples are

single sign-on services and applications that span multiple domains (e.g. `google.com` and `google.be`, etc.). Such a cross-domain interaction pattern shows similar characteristics as a parameter-based session fixation attack, and is also detected and prevented by SERENE's general approach (Figure 2(a)).

From a security point of view, SERENE's general approach is quite effective, since it stops potential session fixation attacks. From the usability point of view however, breaking functionality is to be avoided. Therefore, we slightly relax the general approach for SIDs embedded as a parameter: we allow them to be sent to the server only once, and make sure that they are not used to maintain a potentially fixated session. Practically, SERENE remembers unknown SIDs embedded as a parameter when sending the request, and makes sure these unknown SIDs are removed from the corresponding response if present (e.g. in an element, script, header etc.). This relaxation (Figure 2(b)) allows a valid token to be exchanged, and prevents a fixated SID from doing harm. Even if a fixated SID is sent to the server, it can never be used to establish a fixated session, since SERENE will prevent it to be used on subsequent requests.

4.3 Session Identifier Identification

SERENE's approach depends on the capability to distinguish session identifiers from other cookies and parameters. Earlier approaches to tackle this problem [13,16] are not directly applicable for reasons we discuss in Section 6. Therefore, we used these approaches as inspiration for an elementary SID identification algorithm. Below, we discuss the way the algorithm works. In Section 5, we evaluate the algorithm on existing sites.

The first step in the algorithm is to check whether the key matches an extensive list of 45 known session identifier names. If not, we check whether the key/value pair passes all of the following three heuristics:

1. The key has to include an obvious part of a SID name (e.g. *sess* or *id*)
2. The value has to be sufficiently long (i.e. 10 or more characters).
3. The value has to be sufficiently random [13].

Each of the three heuristics serves a specific purpose. The first heuristic is required to rule out key/value pairs that contain a long, seemingly random value, but are in no way related to a SID. The second heuristic rules out short identifiers, such as product or article identifiers. The third heuristic rules out any non-random values, since they can not serve as a valid session identifier anyway. Any key/value pair not matching these heuristics is either not session-related, or not fit to serve as a session identifier. For instance, a short or non-random value is easily brute forced.

4.4 Prototype Implementation

The prototype of SERENE is implemented as a Firefox add-on, available for any modern version of Firefox. Once SERENE is installed, it protects against session fixation attacks without requiring any configuration.

Due to the extensive add-on support available within the Mozilla framework, the implementation of SERENE is fairly straightforward. Cookie inspection and manipulation is done through the HTTP channel, which provides access to the request before it is sent out and to the response before it is processed. The HTTP channel is also used to detect outgoing SIDs embedded as parameters. Incoming parameters containing SIDs are extracted from the page contents and potentially harmful tokens are removed from the page before it is processed.

5 Evaluation

SERENE's approach for both cookie-based and parameter-based session management successfully counters attack vectors *A1*, *A2*, *A3*, *A4* and a script-based *A6*. Self-reliant client-side protection against header-based attack vectors (*A5* and *A6*) is virtually impossible, since these attack vectors exhibit exactly the same behavioral patterns as the de facto session management techniques used in the majority of modern applications. Due to their damage potential, attacks using attack vector *A5* are scarce. In Section 6, we elaborate on attack vector *A6*.

In the remainder of this section, we evaluate two important aspects that determine the successful applicability of SERENE: (i) does the SID identification algorithm support a large majority of available sites, and (ii) is SERENE compatible with available sites? From a study of Alexa's top one million sites, we show that SERENE already protects 83.43% of analyzed applications, a number that can be increased with future refinements. Additionally, we demonstrate that SERENE fully preserves the functionality of 95.14% of sites.

5.1 Session Identifier Identification

To analyze the coverage of the SID identification algorithm proposed in Section 4.3, we collected the index pages of Alexa's top one million sites, storing both the response headers and the response pages. We used `wget` to collect only the main page, without attempting to load any subresource (e.g. images, scripts, etc.), thus we only sent one request to each listed site. One exception is a response with a redirect, which we followed until it pointed at an actual page.

To assess the validity of the SID identification algorithm, we conducted a manual analysis of a subset of 1,000 sites. This analysis shows that the SID identification algorithm effectively filters out SIDs from other values. Out of 5,500 cookies, 1,953 are identified as SIDs. Of these 1,953 cookies, we only discovered 10 cookies that can not be obviously classified as a SID.

Analyzing the set of top one million sites shows 472,834 sites ask the browser to set a cookie upon the first request. Running the SID identification algorithm on these cookies reveals that the cookies of 349,480 domains (73.98%) contain a session identifier: 266,305 domains use a SID with a known name and 98,305 use a SID that matches the three heuristics. Note that these numbers indicate that 15,130 domains use both a SID with a known name and a SID that matches the heuristics. Manual inspection of a subset shows that several sites use indeed

multiple SID key/value pairs, for instance two different kind of identifiers (a session ID and a visitor ID) or different keys for the same SID value. Finally, analyzing the use of the *Domain* option on cookies containing a session identifier shows that 6.5% of 349,480 sites make the SID available for all subdomains.

These numbers suggest that of the 472,834 sites setting cookies, 123,354 do not include a session identifier in their cookies. We isolated these sites and conducted a follow-up study: similar to the first study, we fetched their index page twice, independently from each other. By comparing the cookie's key/value pairs present in the response, we can detect potential false negatives in the SID identification algorithm: if the cookies of both responses are exactly the same, then these cookies can not represent a session identifier, since a SID can not be shared between two independent requests. The results show that of the 123,354 sites, 77,935 set at least one different cookie value on both requests. Applying the length and randomness heuristic suggests that 69,405 of these domains actually set some kind of identifier. In total, this means that with the elementary SID identification algorithm, SERENE already protects 349,480 domains out of 418,885 domains setting a SID, or 83.43%.

Conclusion The analysis of the support of the elementary SID identification algorithm shows that SERENE is able to protect a large majority of Alexa's top one million sites. In Section 6 we elaborate on potential refinements of the SID identification algorithm, allowing us to increase the level of protection.

5.2 Application Compatibility

In the second part of the evaluation, we take a closer look at the impact of SERENE's protective measures on the functionality of available sites. We prepared a clean Firefox profile with SERENE installed. We instructed Firefox to load each site using this clean profile, stopped Firefox after 25 seconds and collected statistics generated by SERENE. Note that this process not only loads the index pages, but also all included resources, both within the domain and external, thus triggering SERENE's protective measures.

Our study shows that of the one million processed sites, SERENE has no negative effect on the functionality of 524,014 (93.14%) of 562,538 sites that set cookies. A follow-up manual analysis of the most common impacted traffic patterns reveals that third party services, such as tracking, analytics or advertising, often trigger SERENE's protective measures. Several sites have even documented this behavior [3,7]. Additionally, recent initiatives such as tracking protection lists [12] or Do-Not-Track [11] also aim at discouraging this behavior. Removing obvious instances of these services brings SERENE's compatibility to 95.55%.

Conclusion The compatibility study of SERENE's impact on available web applications shows that SERENE fully preserves the functionality of 93.14% of sites. Not counting privacy-invasive third party services brings the level of compatibility up to 95.55%. For the remaining 4.45%, we suggest a follow-up user study to investigate the noticeable impact on an application's functionality.

6 Discussion

Refining SID Identification Earlier work already proposed an algorithm for client-side identification of session identifiers in cookies [13,16]. Both approaches aim at preventing session hijacking attacks, where the attacker steals the cookie containing a SID through JavaScript, allowing him to take over the session. The proposed client-side solution is to attach the *HttpOnly* option to an identified SID, which prevents the cookie from being read from JavaSript. SID detection happens using a selected list of known names, combined with heuristics on the value. Note that in case of false positives, the SID is prevented from being read in JavaScript, but is still sent to the server on outgoing requests.

Initially, we attempted to use such an algorithm for SID identification in SERENE, but the algorithm produced quite a few false positives, both for cookies and for parameters. Unfortunately, a false positive in SERENE means that the value will be removed from the request, so it will never be sent to the server. This effect is severely more disruptive than preventing JavaScript to access a cookie with probably a random SID. We addressed these problems with the elementary SID identification algorithm, as proposed in Section 4 and evaluated in Section 5.

In future work, we suggest to refine the elementary algorithm by carefully integrating the more generic, heuristic algorithms, in order to reduce the false negative rate. To support this suggestion, we ran the 77,935 domains that sent two different cookie values in two independent requests through SessionShield's algorithm, which suggests that further refinement can extend support to 63,384 of these 77,935 domains, resulting in a total compatibility of 98.6%.

Subdomain Attack Vector As mentioned before, SERENE covers all session fixation attack vectors, except for header-based attacks (*A5* and *A6*). Attack vector *A6* is most likely to occur, and is launched through a `Set-Cookie` header that sets a cookie belonging to all subdomains. In order to launch such an attack, the attacker needs to control such a subdomain and needs to be able to set custom response headers (i.e. a `Set-Cookie` header).

Preventing these session fixation attacks at the client-side is currently not possible, because the pattern of an attack is very similar to a legitimate usage pattern, where a domain wants to set a SID belonging to all subdomains. Simply disallowing such a SIDs would break a substantial fraction of sites. Section 5 shows that already 6.5% (22,706 out of 349,480) of sites setting a SID on their index page use the *Domain* option. Bortz et al. [5] also state that existing applications depend on sharing cookies across subdomains.

7 Conclusion

In this paper, we presented SERENE, the first self-reliant client-side countermeasure against session fixation attacks, fully covering 4.5 out of 6 attack vectors. Unfortunately, complete client-side protection is virtually impossible, due to potential abuse of headers, the only legitimate mechanism currently available for

web applications. In an wide-scale study of Alexa's top one million sites, we have shown that SERENE fully preserves 95.14% of functionality, while protecting 83.43% of investigated applications. Future refinement and a follow-up user study are the key to increase both the compatibility and coverage.

References

1. G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh. An analysis of private browsing modes in modern browsers. In *Proceedings of the 19th USENIX conference on Security*, pages 6–6. USENIX Association, 2010.
2. A. Barth, C. Jackson, and J. Mitchell. Securing frame communication in browsers. *Communications of the ACM*, 52(6):83–91, 2009.
3. BBC. Privacy and cookies. `http://www.bbc.co.uk/privacy/`, 2012.
4. B. Bonné. Improving session security in web applications. `http://research.edm.uhasselt.be/~bbonne/docs/Thesis.pdf`.
5. A. Bortz, A. Barth, and A. Czeskis. Origin cookies: Session integrity for web applications. 2011.
6. P. De Ryck, L. Desmet, W. Joosen, and F. Piessens. Automatic and precise client-side protection against csrf attacks. *ESORICS 2011*, pages 100–116, 2011.
7. Delia Online. Cookies used on delia online. `http://www.deliaonline.com/home/delia-online-cookies.html`, 2012.
8. M. Johns, B. Braun, M. Schrank, and J. Posegga. Reliable Protection Against Session Fixation Attacks. In *Proceedings of the 26th ACM Symposium on Applied Computing (SAC)*, 2011.
9. E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 330–337. ACM, 2006.
10. C. Linhart, A. Klein, R. Heled, and S. Orrin. Http request smuggling. *Computer Security Journal*, 22(1):13, 2006.
11. J. Mayer and A. Narayanan. Do not track - universal web tracking opt out. `http://donottrack.us/`, 2011.
12. Microsoft Corporation. Tracking protection lists. `http://ie.microsoft.com/testdrive/Browser/TrackingProtectionLists/`, 2011.
13. N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen. SessionShield: Lightweight Protection against Session Hijacking. In *Proceedings of the 3rd International Symposium on Engineering Secure Software and Systems (ESSoS)*, 2011.
14. J. Samuel. Requestpolicy 0.5.20. `http://www.requestpolicy.com`, 2011.
15. M. Schrank, B. Braun, M. Johns, and J. Posegga. Session Fixation - the Forgotten Vulnerability? In *Proceedings of the 5th conference on "Sicherheit, Schutz und Zuverlssigkeit" (GI Sicherheit 2010)*, 2010.
16. S. Tang, N. Dautenhahn, and S. T. King. Fortifying web-based applications automatically. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2011.
17. M. Ter Louw, K. T. Ganesh, and V. N. Venkatakrishnan. Adjail: Practical enforcement of confidentiality and integrity policies on web advertisements. In *19th USENIX Security Symposium*, 2010.
18. J. Williams and D. Wichers. Owasp top 10. *OWASP Foundation*, 2010.
19. Y. Zhou and D. Evans. Why Aren't HTTP-only Cookies More Widely Deployed? In *Proceedings of 4th Web 2.0 Security and Privacy Workshop (W2SP '10)*, 2010.